UNIT III

Unit-3 Coding & Testing

- Programming Practice
- Testing Fundamentals (errors, fault & failure)
- Levels of Testing
- Testing Methods

- **Programming Practice**

Coding Standards in Software Engineering

Some of the coding standards are given below:

1. **Limited use of globals:** These rules tell about which types of data that can be declared global and the data that can't be.
2. **Standard headers for different modules:** For better understanding and maintenance of the code, the header of different modules should follow some standard format and information. The header format must contain below things that is being used in various companies:
   - Name of the module
   - Date of module creation
   - Author of the module
   - Modification history
   - Synopsis of the module about what the module does
   - Different functions supported in the module along with their input output parameters
   - Global variables accessed or modified by the module
3. **Naming conventions for local variables, global variables, constants and functions:** Some of the naming conventions are given below:
   - Meaningful and understandable variables name helps anyone to understand the reason of using it.
   - Local variables should be named using camel case lettering starting with small letter (e.g. **localData**) whereas Global variables names should start with a capital letter (e.g. **GlobalData**). Constant names should be formed using capital letters only (e.g. **CONSDATA**).
   - It is better to avoid the use of digits in variable names.
   - The names of the function should be written in camel case starting with small letters.
   - The name of the function must describe the reason of using the function clearly and briefly.
4. **Indentation:** Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:
   - There must be a space after giving a comma between two function arguments.
   - Each nested block should be properly indented and spaced.
   - Proper Indentation should be there at the beginning and at the end of each block in the program.
   - All braces should start from a new line and the code following the end of braces also start from a new line.
5. **Error return values and exception handling conventions:** All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging.

Coding Guidelines in Software Engineering

Coding guidelines give some general suggestions regarding the coding style that to be followed for the betterment of understandability and readability of the code.
Some of the coding guidelines are given below:

1. **Avoid using a coding style that is too difficult to understand:** Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.
2. **Avoid using an identifier for multiple purposes:** Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.
3. **Code should be well documented:** The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.
4. **Length of functions should not be very large:** Lengthy functions are very difficult to understand. That's why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.
5. **Try not to use GOTO statement:** GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.

Advantages of Coding Guidelines

1. Coding guidelines increase the efficiency of the software and reduces the development time.
2. Coding guidelines help in detecting errors in the early phases, so it helps to reduce the extra cost incurred by the software project.
3. If coding guidelines are maintained properly, then the software code increases readability and understandability thus it reduces the complexity of the code.
4. It reduces the hidden cost for developing the software.

Software Testing - Bug vs Defect vs Error vs Fault vs Failure

Software Testing defines a set of procedures and methods that check whether the actual software product matches with expected requirements, thereby ensuring that the product is Defect free. There are a set of procedures that needs to be in mind while testing the software manually or by using automated procedures. The main purpose of software testing is to identify errors, deficiencies, or missing requirements with respect to actual requirements. Software Testing is Important because if there are any bugs or errors in the software, they can be identified early and can be solved before the delivery of the software product. The article focuses on discussing the difference between bug, defect, error, fault, and failure.

What is a Bug?

A bug refers to defects which means that the software product or the application is not working as per the adhered requirements set. When we have any type of logical error, it causes our code to break, which results in a bug. It is now that the Automation/ Manual Test Engineers describe this situation as a bug.

- A bug once detected can be reproduced with the help of standard bug-reporting templates.
- Major bugs are treated as prioritized and urgent especially when there is a risk of user dissatisfaction.
- The most common type of bug is a crash.
- Typos are also bugs that seem tiny but are capable of creating disastrous results.

What is a Defect?

A defect refers to a situation when the application is not working as per the requirement and the actual and expected result of the application or software are not in sync with each other.

- The defect is an issue in application coding that can affect the whole program.
- It represents the efficiency and inability of the application to meet the criteria and prevent the software from performing the desired work.
- The defect can arise when a developer makes major or minor mistakes during the development phase.

What is an Error?

Error is a situation that happens when the Development team or the developer fails to understand a requirement definition and hence that misunderstanding gets translated into buggy code. This situation is referred to as an Error and is mainly a term coined by the developers.

- Errors are generated due to wrong logic, syntax, or loop that can impact the end-user experience.
- It is calculated by differentiating between the expected results and the actual results.
- It raises due to several reasons like design issues, coding issues, or system specification issues and leads to issues in the application.

What is a Fault?

Sometimes due to certain factors such as Lack of resources or not following proper steps Fault occurs in software which means that the logic was not incorporated to handle the errors in the application. This is an undesirable situation, but it mainly happens due to invalid documented steps or a lack of data definitions.

- It is an unintended behavior by an application program.
- It causes a warning in the program.
- If a fault is left untreated it may lead to failure in the working of the deployed code.
- A minor fault in some cases may lead to high-end error.

- There are several ways to prevent faults like adopting programming techniques, development methodologies, peer review, and code analysis.

What is a Failure?

Failure is the accumulation of several defects that ultimately lead to Software failure and results in the loss of information in critical modules thereby making the system unresponsive. Generally, such situations happen very rarely because before releasing a product all possible scenarios and test cases for the code are simulated. Failure is detected by end-users once they face a particular issue in the software.

- Failure can happen due to human errors or can also be caused intentionally in the system by an individual.
- It is a term that comes after the production stage of the software.
- It can be identified in the application when the defective part is executed.

A simple diagram depicting Bug vs Defect vs Fault vs Failure:

What is Software Testing?

Software Testing is a process of verifying and validating whether the **Software Product** or **Application** is working as expected or not. The complete testing includes identifying errors and bugs that cause future problems for the performance of an application.

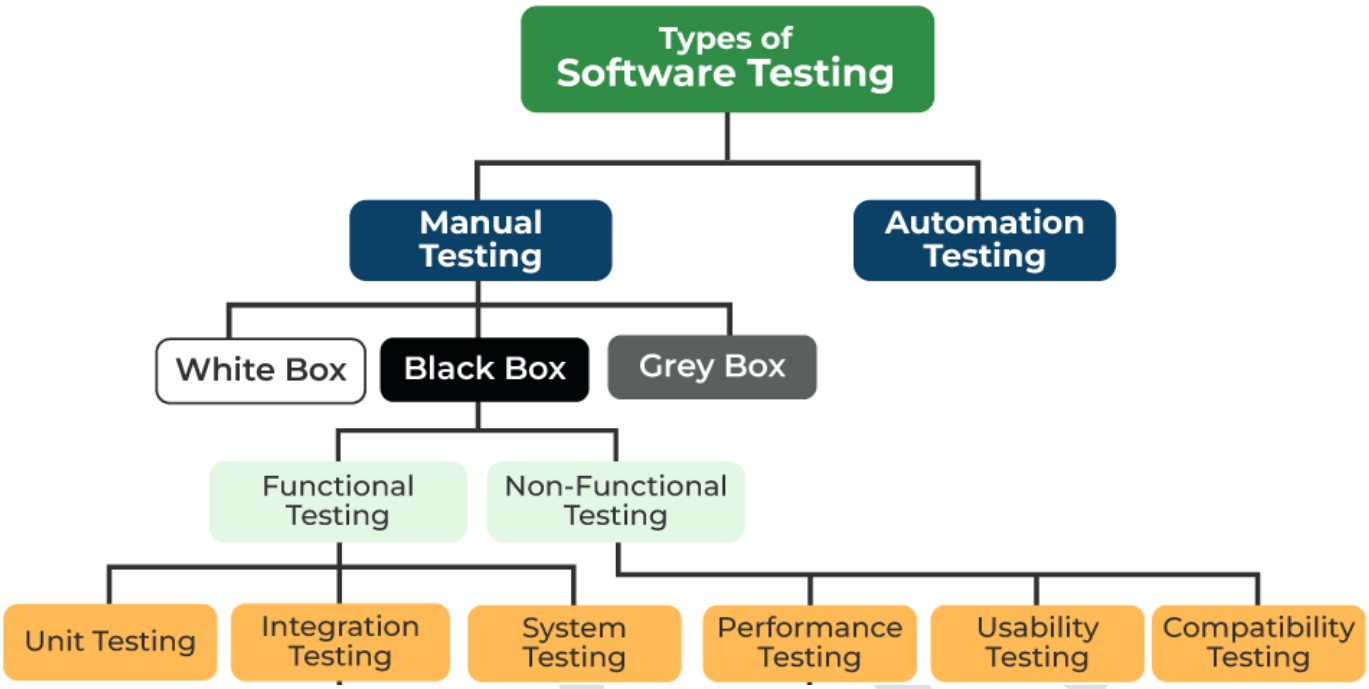
Software Testing Can be Divided into Two Steps:

Software testing mainly divides into the two parts, which is used in the **Software Development Process:**

1. **Verification**: This step involves checking if the software is doing what is supposed to do. Its like asking, "**Are we building the product the right way?**"
2. **Validation**: This step verifies that the software actually meets the customer's needs and requirements. Its like asking, "**Are we building the right product?**"

Types of Software Testing

Here are the **Types of Software Testing** mainly categorized into the two domain, which are below.



1. Manual Testing

Manual Testing is a technique to test the software that is carried out using the functions and features of an application. Which means manual testing will check the defect manually with trying one by one function is working as expected.

2. Automation Testing

Automation Testing It is a technique where the Tester writes scripts independently and uses suitable Software or Automation Tools to test the software. It is an Automation Process of a Manual Process. It allows for executing repetitive tasks without the use of a Manual Tester.

Types of Manual Testing

Manual testing will be divided into further types which is following:

1. White Box Testing

White Box Testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

2. Black Box Testing

Black-Box Testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements.

3. Gray Box Testing

Gray Box Testing is a software testing technique that is a combination of the **Black Box Testing** technique and the **White Box Testing** technique. In the Black Box Testing technique, the tester is unaware of the internal structure of the item being tested and in White Box Testing the internal structure is known to the tester.

Types of Black Box Testing

Black Box Testing will be divided into further types which is following:

1. Functional Testing

Functional Testing is a type of Software Testing in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application.

2. Non-Functional Testing

Non-Functional Testing is a type of Software Testing that is performed to verify the non-functional requirements of the application. It verifies whether the behavior of the system is as per the requirement or not. It tests all the aspects that are not tested in functional testing.

Types of Functional Testing

Functional Testing will be divided into further types which is following:

1. Unit Testing

Unit Testing is a method of testing individual units or components of a software application. It is typically done by developers and is used to ensure that the individual units of the software are working as intended.

2. Integration Testing

Integration Testing is a method of testing how different units or components of a software application interact with each other. It is used to identify and resolve any issues that may arise when different units of the software are combined.

3. System Testing

System Testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users.

Types of Non-functional Testing

Here are the Types of Non-Functional Testing

1. Performance Testing

Performance Testing is a type of software testing that ensures software applications perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity, and stability under a particular workload.

2. Usability Testing

Usability Testing in software testing is a type of testing, that is done from an end user's perspective to determine if the system is easily usable. Usability testing is generally the practice of testing how easy a design is to use on a group of representative users.

3. Compatibility Testing

Compatibility Testing is software testing that comes under the **nonfunctional testing** category, and it is performed on an application to check its compatibility (running capability) on different platforms/environments. This testing is done only when the application becomes stable.

*You can learn more **Software Testing Types** which is included in the **Software testing** in detail.*

Different Levels of Software Testing

In Software testing there are **Different Levels of Testing** can be majorly classified into 4 levels:

1. **Unit Testing:** In this type of testing, errors are detected individually from every component or unit by individually testing the components or units of software to ensure that they are fit for use by the developers. It is the smallest testable part of the software.
2. **Integration Testing:** In this testing, two or more modules which are unit tested are integrated to test i.e., technique interacting components, and are then verified if these integrated modules work as per the expectation or not, and interface errors are also detected.
3. **System Testing:** In system testing, complete and integrated Software are tested i.e., all the system elements forming the system are tested as a whole to meet the requirements of the system.

4. **Acceptance Testing:** This is a kind of testing conducted to ensure that the requirements of the users are fulfilled before its delivery and that the software works correctly in the user's working environment.

Testing methods

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	**Disadvantages**
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
Clearly separates user's perspective from the developer's perspective through visibly defined roles.	Blind coverage, since the tester cannot target specific code segments or error prone areas.
Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	The test cases are difficult to design.

White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	**Disadvantages**
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
It helps in optimizing the code.	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.
Extra lines of code can be removed which can bring in hidden defects.	It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the

database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	**Disadvantages**
Offers combined benefits of black-box and white-box testing wherever possible.	Since the access to source code is not available, the ability to go over the code and test coverage is limited.
Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.	The tests can be redundant if the software designer has already run a test case.
Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.	Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.